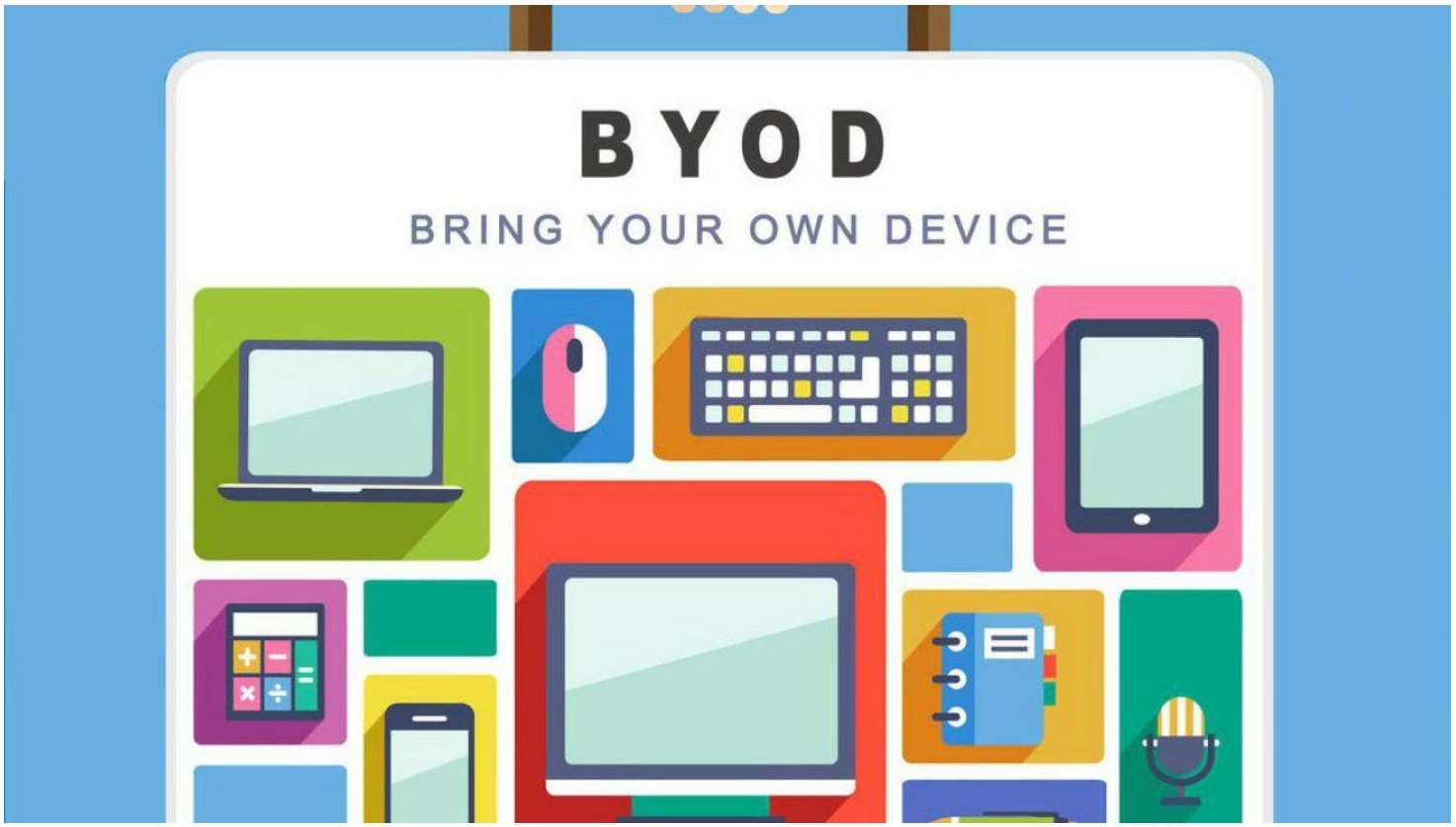
PAI3. BYODSEC-BRING YOUR OWN DEVICE SEGURO PARA UNA EMPRESA USANDO SSL/TLS



Grupo 9
Asier Herrería Oña
Jaime Cortés Vázquez
Fco. Javier Viera Chaves
Víctor Manuel Vázquez García

ÍNDICE

INTRODUCCIÓN

En este proyecto de aseguramiento de la información se solicita la implantación de un sistema con el que se pueda aplicar la política de seguridad Bring Your Own Device (BYOD) de forma segura.

Dicha política se basa en la utilización de los dispositivos de los propios empleados, con los que pueden acceder a distintos recursos pertenecientes a la empresa como correos electrónicos, bases de datos y archivos en servidores corporativos.

Para poder realizar transmisiones seguras de todos estos elementos es necesario implementar canales de comunicación seguros, por tanto se hará uso de Sockets (especificación de una dirección IP y un puerto) del tipo SSL (Secure Sockets Layers) que permite la creación de estos para poder implementar los requisitos de autenticidad, confidencialidad e integridad solicitados.

OBJETIVOS

Para la realización del sistema BYOD presentado previamente haremos uso de Sockets que implementarán el protocolo SSL/TLS que proporciona la capa de autenticidad, confidencialidad e integridad requeridos. Con el objetivo de satisfacer la demanda propuesta se plantean las siguientes soluciones:

- Desarrollar/seleccionar como llevar a la práctica los canales de comunicación segura para la transmisión de credenciales (usuario, contraseñas) y un mensaje con el Protocolo SSL/TLS (autenticidad, confidencialidad e integridad).
- Utilizar una herramienta de análisis de tráfico que permita comprobar la confidencialidad e integridad de los canales de comunicaciones seguros.
- Analizar la seguridad de los Cipher Suites usados y el funcionamiento de estos dentro del protocolo SSL/TLS en sus diferentes versiones.

DESARROLLO

TAREA 1: Generación de la infraestructura Cliente-Servidor

- **A**. Creación de un keyStore – Compartición de claves

  Para el protocolo SSL/TLS es necesario un certificado de autenticación autofirmado. Para ello se realizarán los siguientes pasos:

  - En primer lugar hay que realizar la instalación de SSL en el equipo:

    ```
    sudo apt-get install openssl
    ```

  - Después, se creará una clave privada, que será útil para la generación del certificado:

    ```
    openssl genrsa -out server.key 1024
    ```

  - Lo siguiente será la creación de un CSR (Certificate Signing Request), que es la base del certificado, donde se especifica la región, provincia, localidad, dominio, nombre, email...

    ```
    openssl req -new -key server.key -out server.csr
    ```

- Por último, se generará el certificado SSL. Para ello, se hace uso de la clave y del CSR creado anteriormente. El parámetro **days** sirve para definir la fecha de expiración del certificado.

```
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Una vez completados todos los pasos explicados anteriormente se podrá hacer uso de SSL.

- **B**. Generación de un Socket TLS Servidor

  Para generar la comunicación entre servidor y cliente, se utilizará lenguaje Python. Se ha diseñado un script que actúa como servidor, su función será mantenerse a la escucha y establecer conexión con el cliente. Una vez establecida la conexión y negociado los parámetros del cifrado recibirá el mensaje por el cliente y lo desencriptará usando la clave simétrica negociada anteriormente.

```
import socket, ssl

bindsocket = socket.socket()
bindsocket.bind(('localhost', 10035))
bindsocket.listen(5)

while True:
    newsocket, fromaddr = bindsocket.accept()
    connstream = ssl.wrap_socket(newsocket,
                                 server_side=True,
                                 certfile="server.crt",
                                 keyfile="server.key",
                                 ciphers = "ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:AES128-GCM-SHA256:AES128-SHA256:HIGH:")
    print "Recibido del cliente: ", connstream.read()
    connstream.shutdown(socket.SHUT_RDWR)
    connstream.close()
```

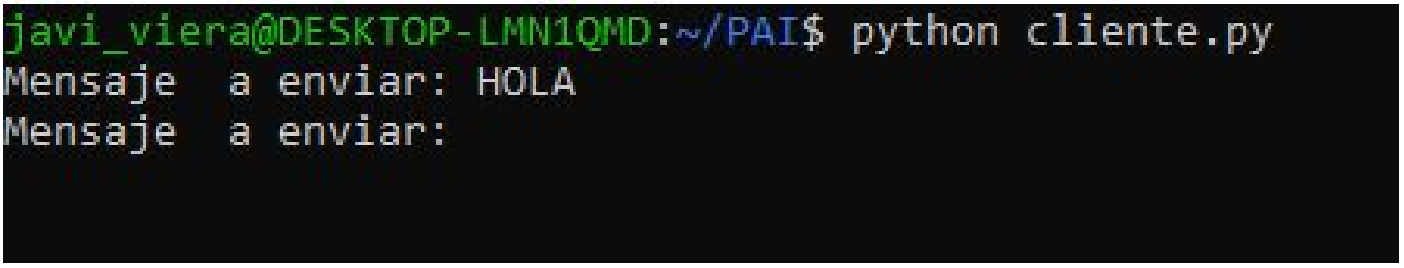
- **C**. Generación de un Socket TLS Cliente

  En la parte del cliente se ha generado otro script, su función será establecer conexión con el servidor y decirle cómo será la clave usada en el envío del mensaje. Una vez establecida se procederá al envío del mensaje cifrado.

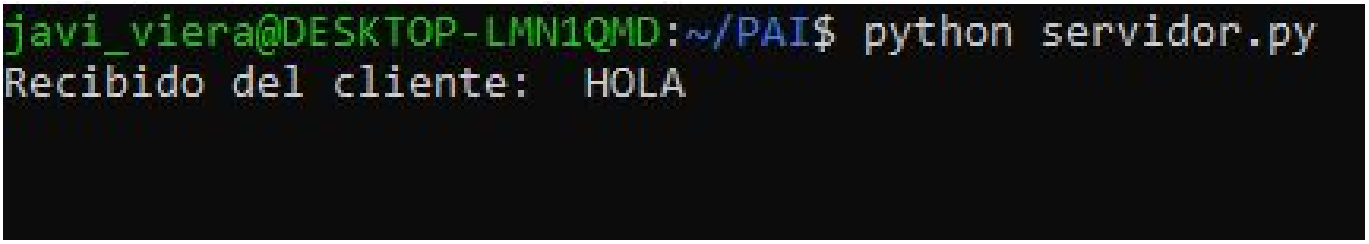
```
import socket, ssl

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Require a certificate from the server. We used a self-signed certificate
# so here ca_certs must be the server certificate itself.
ssl_sock = ssl.wrap_socket(s,
                           ca_certs="server.crt",
                           cert_reqs=ssl.CERT_REQUIRED,
                           ciphers = "ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:AES128-GCM-SHA256:AES128-SHA256:HIGH:")

ssl_sock.connect(('localhost', 10035))

while  True:
    mensaje = raw_input("Mensaje  a enviar: ")

#invoco  el metodo send pasando como parametro el string ingresado por el  usuario
    ssl_sock.write(mensaje)
    ssl_sock.close()
```

A continuación se muestran algunas capturas de pantalla del correcto funcionamiento de los scripts generados para el cliente y el servidor respectivamente:

- **CLIENTE**



- **SERVIDOR**



TAREA 2: Análisis de tráfico de red en comunicaciones (sniffers)

- **Wireshark.**

  Se han realizado capturas de tráfico de la comunicación entre el cliente y el servidor de manera directa con Wireshark.
  Al abrir el programa, seleccionamos la interfaz loopback, ya que esta comunicación se realiza en el **localhost** mediante un puerto que hemos determinado. Para ver que esta comunicación se realiza correctamente, tomaremos el código de cliente y servidor de la práctica de verificadores de integridad en la transmisión de punto a punto de información (PAI 2), para ver que si realizamos captura de tráfico de esta comunicación podemos ver la información que se pasa desde el cliente al servidor:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	52378 → 8050 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
2	0.000081	127.0.0.1	127.0.0.1	TCP	56	8050 → 52378 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
3	0.000142	127.0.0.1	127.0.0.1	TCP	44	52378 → 8050 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	2.034781	127.0.0.1	127.0.0.1	TCP	221	52378 → 8050 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=177
5	2.034816	127.0.0.1	127.0.0.1	TCP	44	8050 → 52378 [ACK] Seq=1 Ack=178 Win=2619392 Len=0
6	2.047033	127.0.0.1	127.0.0.1	TCP	221	8050 → 52378 [PSH, ACK] Seq=1 Ack=178 Win=2619392 Len=177
7	2.047067	127.0.0.1	127.0.0.1	TCP	44	52378 → 8050 [ACK] Seq=178 Ack=178 Win=2619392 Len=0
8	3.546373	127.0.0.1	127.0.0.1	TCP	218	52378 → 8050 [PSH, ACK] Seq=178 Ack=178 Win=2619392 Len=174
9	3.546408	127.0.0.1	127.0.0.1	TCP	44	8050 → 52378 [ACK] Seq=178 Ack=352 Win=2619392 Len=0
10	3.558512	127.0.0.1	127.0.0.1	TCP	218	8050 → 52378 [PSH, ACK] Seq=178 Ack=352 Win=2619392 Len=174
11	3.558546	127.0.0.1	127.0.0.1	TCP	44	52378 → 8050 [ACK] Seq=352 Ack=352 Win=2619392 Len=0

Wireshark · Follow TCP Stream (tcp.stream eq 0) · Captura sin SSL.pcapng

```
564b96ec7b570bb6ff696b3bbac8a594a13094edcb47f9eb63d008a8f896a2f8c6283c8165267fde71c721b1cab036c0ce80a5070081716a85644274ab3e8b3fLa
fecha y hora de envio: 16/11/19 17:21:20:
hola564b96ec7b570bb6ff696b3bbac8a594a13094edcb47f9eb63d008a8f896a2f8c6283c8165267fde71c721b1cab036c0ce80a5070081716a85644274ab3e8b3f
La fecha y hora de envio: 16/11/19 17:21:20:
holab170bcf593463526e7c9104549c89e0e400913a159490cc7590863992a88342e3a066827c7ae9308449cbe9577625b11e6fbde2e02e83c431d2200602ca8ff39
La fecha y hora de envio: 16/11/19 17:21:21:
lb170bcf593463526e7c9104549c89e0e400913a159490cc7590863992a88342e3a066827c7ae9308449cbe9577625b11e6fbde2e02e83c431d2200602ca8ff39La
fecha y hora de envio: 16/11/19 17:21:21: l
```

Ahora mostraremos capturas de tráfico de la comunicación segura (SSL) implementada:

No.	Time	Source	Destination	Protocol	Length	Info
5	3.815472	127.0.0.1	127.0.0.1	TCP	56	52238 → 10035 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
6	3.815555	127.0.0.1	127.0.0.1	TCP	56	10035 → 52238 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
7	3.815624	127.0.0.1	127.0.0.1	TCP	44	52238 → 10035 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
8	3.816499	127.0.0.1	127.0.0.1	TLSv1.3	561	Client Hello
9	3.816537	127.0.0.1	127.0.0.1	TCP	44	10035 → 52238 [ACK] Seq=1 Ack=518 Win=2619136 Len=0
10	3.824429	127.0.0.1	127.0.0.1	TLSv1.3	1057	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
11	3.824482	127.0.0.1	127.0.0.1	TCP	44	52238 → 10035 [ACK] Seq=518 Ack=1014 Win=2618624 Len=0
12	3.826824	127.0.0.1	127.0.0.1	TLSv1.3	124	Change Cipher Spec, Application Data
13	3.826871	127.0.0.1	127.0.0.1	TCP	44	10035 → 52238 [ACK] Seq=1014 Ack=598 Win=2619136 Len=0
14	3.827247	127.0.0.1	127.0.0.1	TLSv1.3	299	Application Data
15	3.827292	127.0.0.1	127.0.0.1	TCP	44	52238 → 10035 [ACK] Seq=598 Ack=1269 Win=2618368 Len=0
16	3.827424	127.0.0.1	127.0.0.1	TLSv1.3	299	Application Data
17	3.827456	127.0.0.1	127.0.0.1	TCP	44	52238 → 10035 [ACK] Seq=598 Ack=1524 Win=2618112 Len=0
18	6.126635	127.0.0.1	127.0.0.1	TLSv1.3	70	Application Data
19	6.126674	127.0.0.1	127.0.0.1	TCP	44	10035 → 52238 [ACK] Seq=1524 Ack=624 Win=2619136 Len=0
20	6.128315	127.0.0.1	127.0.0.1	TCP	44	10035 → 52238 [FIN, ACK] Seq=1524 Ack=624 Win=2619136 Len=0
21	6.128349	127.0.0.1	127.0.0.1	TCP	44	52238 → 10035 [ACK] Seq=624 Ack=1525 Win=2618112 Len=0

Wireshark · Follow TCP Stream (tcp.stream eq 1) · Captura Cliente-Servidor.pcapng

```
...........C.O....N....g.:.=!...~.E.
?u<.; ..b-/U.h.^n[....);..
t.."Sy.^.O..........+./...<.,.0...................].a.W.S.............\.`.V.R.$.(.k.j.s.w.....#.'.g.@.r.v.....
...9.8..... ...3.2.E.D.......Q.....P.=.....5.../.A..............
...
...........#...........
.0.............        .
.................................+.           ..........-.....3.&.$... .7*.+ko.!.T.y;.
NVBgQ..|.......y.....................................................................................................
.............................z...v..:y.(..n.3. .L+@.TI..J2...}Z.W..3 ..b-/U.h.^n[....);..
t.."Sy.^.O.......+.....3.$... .r.3.m...S*..v....2....Cv..*...O............G>rB...!|....aP...d.Nf....g..,..[.....
.9.....N?Y.....0...Y.0...........e..#.......)...h.gJ.M.l...S.
..T.\.z.....,..X. ..h$*$...<.j..E..p.hV5...;5t.9w.zo.....'1...5.@S./.-bM3...\/...B.9.u...M=<....k:..   .fQ>...l}8j....o.
73......]..j..E.....&_X.....`.\.....-...+.W..
.>3....%..f...q...,>..&7..Y-d2..Z-.5.....J`.....k...A...=a..99*>...H..          (......m...!q.....A.c;z.          f:..!g....
{.....?X../.{.Q
|../...."..KBj@-.6_6.Q6....P.!.t./..          3.i....._..9......ou3.l....B...~..+.Y....(.......9........&$..IEO.....M }.&.
%K..>.~..v....D.9".]......t.....p..k......h.b...Q..z.q|q7...'.V.e.......+7+D..U...x.J.}X.......xs...g\%..;..C&U.
]#...?....f]A=No...-.f<.c.......L.....1(..%.^bqN...r......;.....X...p\....)1D8
.M...}fb......72<#.Gp...\.ND.0>..P=..!,...+....y.VI`.`..W51..9m.&)..='3.O..fm..+..{...h...]S..!E.....H........E>.-.,wz..l.z.-
U.....m...0.8.....I.m.#."......&.vx..;4....$%Z.tMG.6.............ES...r......
.Y....w.../..zVC...#*QM.. .w.l...............[.X.S..O.v] .....@;..."...w...^.6,...C{....'..DsX...h.m.c.....).x9k..Zu'+..
[,.......I..!.U
:B..n
.gbrT....eQ#........Rh...ff..vI.....V..d.Wq.."\..''l..&.......\....B..e.........../oAw....N...#..-.x?
2.^...~..]..........P..aH..'.......F.......8.....(.....]!.....S....b........`..,..GY..}.o....KD...._...i.\0...k;.R.
9f.g|...w...d..wS)9.d..t6.u.......D.r.i.TK._U..          ..6.J2u..b.z.J....a...f.B.T.:..#...j.r.........l..HZ.6.F.(.T..
0....j@K-.r.._.|.,....B1pQ.1.E...#...]A..^.f.....AQ2!.M..Y.V.#m...f#..H'C..^.3.TD..|.\k.^.ef.......-.....?..4tQ..s.~^.....Y6..
```

Se puede observar que con la implementación del protocolo TLS la información se envía cifrada por lo que no se puede interceptar el mensaje, el usuario y la contraseña.
Viendo el protocolo TLS, en una de sus opciones observamos que el keyStore es diferente tanto para cliente como para el servidor.

TAREA 3: Selección e implementación del conjunto de cipher suite más adecuado para la seguridad con SSL/TLS:

A la hora de realizar la comunicación entre cliente y servidor podemos observar en la captura de tráfico de la trama del cliente (utilizando TLS-v1.2, también soportado por TLS-v1.3) como ofrece una lista extensa de cipher suites posibles a utilizar (clickando en el protocolo TLS->HandShake Protocol:Client Hello->Cipher Suites), eligiendo el servidor el más conveniente para su uso.

```
Cipher Suites (75 suites)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256 (0x003c)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_DHE_DSS_WITH_AES_256_GCM_SHA384 (0x00a3)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
    Cipher Suite: TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xccaa)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8 (0xc0af)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CCM (0xc0ad)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CCM_8 (0xc0a3)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CCM (0xc09f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_ARIA_256_GCM_SHA384 (0xc05d)
    Cipher Suite: TLS_ECDHE_RSA_WITH_ARIA_256_GCM_SHA384 (0xc061)
    Cipher Suite: TLS_DHE_DSS_WITH_ARIA_256_GCM_SHA384 (0xc057)
    Cipher Suite: TLS_DHE_RSA_WITH_ARIA_256_GCM_SHA384 (0xc053)
    Cipher Suite: TLS_DHE_DSS_WITH_AES_128_GCM_SHA256 (0x00a2)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 (0xc0ae)
```

En este caso, el servidor ha utilizado TLS_AES_256_GCM_SHA384 (especificado en el código del servidor), donde el tipo de cifrado (AES) es el más seguro actualmente y podemos decir que es el cipher suite más adecuado para la comunicación.